Exámen Fullstack - FizzBuzz

Diseño y Desarrollo del Backend

- Intraway -

Pablo Méndez

Abril 2022

Análisis de requerimientos

Los requerimientos de la aplicación a desarrollar son los siguientes:

- Generar una API REST que resuelva el problema de encontrar todos los números enteros existentes entre otros dos enteros dados, a saber MIN y MAX.
    - Donde MIN es el primer parámetro que se obtiene del path y MAX el segundo.

- Se debe controlar que MIN sea efectivamente menor o igual a MAX.

- Cualquiera de los números enteros descubiertos debe ser reemplazado por un texto, a saber Fizz, Buzz, FizzBuzz, bajo estas consideraciones:
    - Si el entero descubierto es múltiplo de 3 se debe reemplazar por el texto Fizz.
    - Si el entero descubierto es múltiplo de 5 se debe reemplazar por el texto Buzz.
    - Si el entero descubierto es múltiplo de 3 y de 5 a la misma vez se debe reemplazar por el texto FizzBuzz.

- La respuesta de un request correcto debe ser en formato Json, se muestra a continuación una respuesta de este tipo con resultados :
    - {"timestamp":"1650804958901","code":"143","description":"se encontraron múltiplos de 3 y de 5","list":"8,Fizz,Buzz,11,Fizz,13,14,FizzBuzz"}

- La respuesta de un request incorrecto debe ser en formato Json, se muestra a continuación una respuesta de este tipo con resultados :
    - {"timestamp":"1650805150319","status":400,"error":"Bad Request","exception":"com.intraway.exceptions.badrequest","message":"Los parámetros enviados son incorrectos","path":"/intraway/api/fizzbuzz/80/15"}

- Ejemplos de request correctos:
    - /intraway/api/fizzbuzz/8/15
    - /intraway/api/fizzbuzz/-8/-3
    - /intraway/api/fizzbuzz/-8/15

- Ejemplo de request incorrectos:
    - /intraway/api/fizzbuzz/8/-15
    - /intraway/api/fizzbuzz/-80/-15
    - /intraway/api/fizzbuzz/8.5/15
    - /intraway/api/fizzbuzz/8,5/15

- Los datos resultados deben persistir.

Determinaciones asumidas del negocio

Se asumen determinadas características funcionales, estas son:

1. Los Request exigen siempre dos variables de entrada en el path, (a saber MIN y MAX).
2. Los String Fizz, Buzz y FizzBuzz son cargados desde el archivo application.yml
3. Las respuestas siempre tienen los mismos parámetros, no cambia el formato del objeto JSON devuelto.
    a. La respuesta de un Request correcto tiene un valor “code” que es generado por un Random en tiempo de ejecución.
    b. La respuesta de un Request erróneo tiene siempre el mismo valor en los atributos: “status”, “error”, “exception”, “message”. Solo varían su valor los atributos: “timestamp” y “path”.
4. Por la definición del punto 3.b se decide no generar una tabla para almacenar las respuestas de Request erróneo, este caso se identifica con un campo “STATE” en la tabla “INVOCATIONS”, donde el valor 0 indica una respuesta de Request erróneo y el valor 1 indica una respuesta de Request correcto.

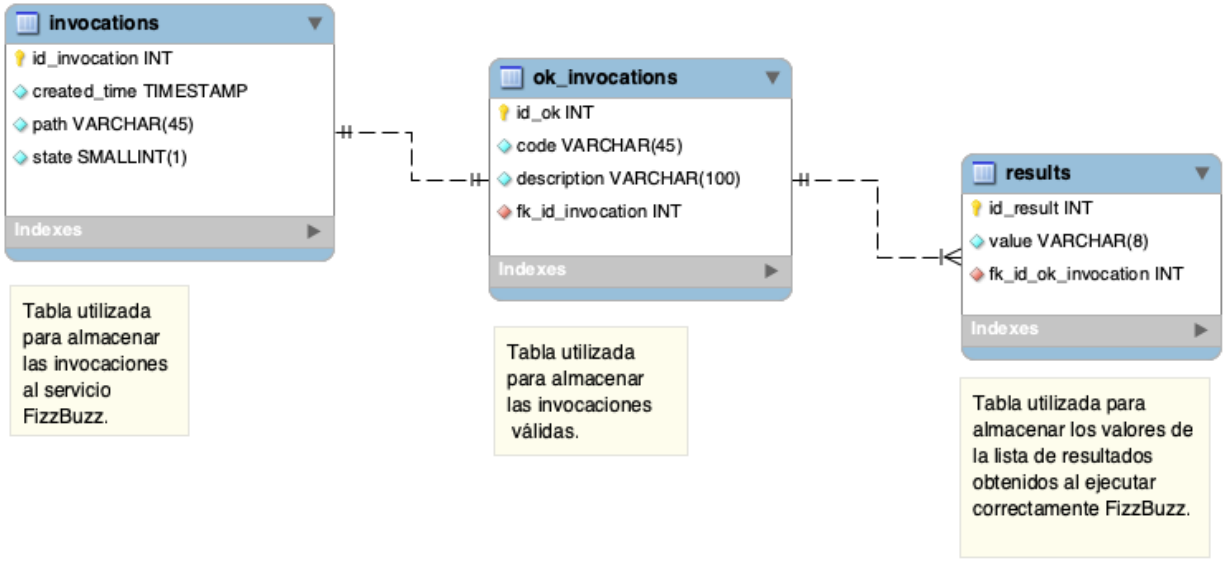
Modelo de datos

Análisis del problema de almacenamiento de datos

Debido a que los datos que deben almacenarse tienen una estructura fija, se decide utilizar una base de datos relacional, y como la experiencia en MySql es mayor, es el motor que se selecciona para cumplir este requisito.

Con respecto al diseño del modelo de datos, una de las opciones es utilizar para el array de resultados un campo JSON, soportado por versiones de MySql posteriores a 5.7.8, pero para estandarizar el modelo, y que sea soportado por un motor distinto de base de datos relacionales, se crea una tabla específica para almacenar los resultados obtenidos al ejecutar el algoritmo de fizzbuzz.

De esta forma el modelo de datos queda según la siguiente imagen. (Se utiliza MySql Workbench para diagramar y sincronizar los cambios en la base de datos).



Análisis del Modelo de Datos definido

El diseño del MER consta de tres tablas. Estas son:

- invocations: Tabla utilizada para almacenar las invocaciones al servicio FizzBuzz. A su vez tiene estos campos:
    - id_invocation: campo del identificador único del registro.
    - created_time: indica fecha de creación del registro (fecha de ejecución de la solicitud).
    - url: utilizado para guardar el cabezal del call al servicio.
    - state: indica si la operación se ejecutó con éxito obteniendo un resultado válido (state = 1), o por el contrario existe un error al invocar el servicio (state = 0).

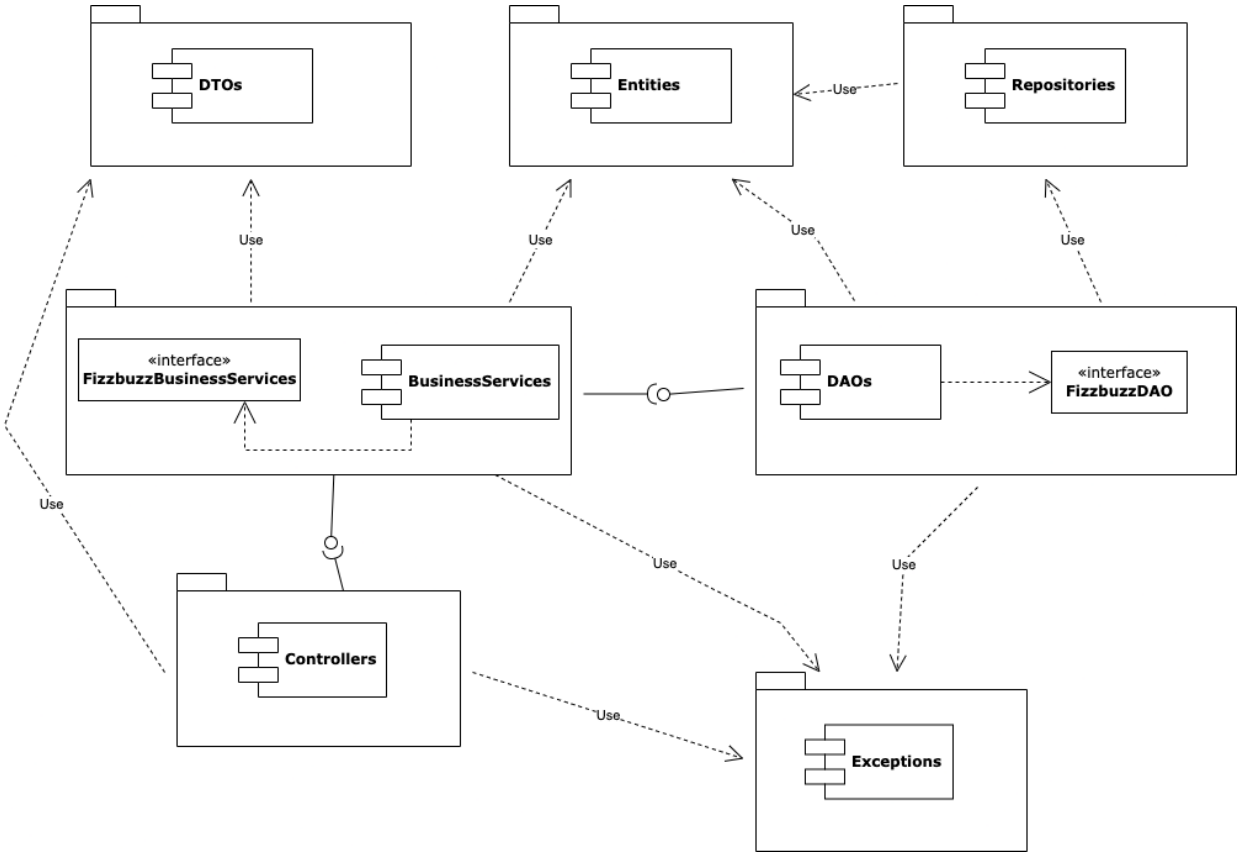
- ok_invocations: Tabla utilizada para almacenar las invocaciones al servicio que resultaron exitosas. Estos son sus campos:
    - id_ok: campo del identificador único del registro.
    - code: código generado luego de la invocación al servicio..
    - description: descripción del resultado de la invocación al servicio.
    - fk_id_invocation: clave foránea que identifica la invocación que generó el resultado.

- results:Tabla utilizada para almacenar los resultados descubiertos para una Request correcto. Los campos que tiene son:
    - id_result: campo con identificador único del registro.
    - value: valor descubierto por el algoritmo FizBuzz
    - fk_id_ok_invocation: clave foránea que identifica la invocación que generó el resultado.

Determinaciones de diseño

Componentes principales de la aplicación

El diagrama a continuación muestra cómo se relacionan los principales componentes implementados en la aplicación.



A continuación se detallan los componentes principales diseñados en la aplicación:

- **BusinessServices**, es el componente donde se implementa la lógica del negocio, haciendo uso de la información recibida desde un request y la información almacenada en base de datos. Utiliza objetos DAO, Entities y DTO.

- **DAO**, es el componente donde se implementa la lógica de acceso a datos, haciendo uso del componente Repositories y de Entities.

- **Repositories**, es el componente que agrupa los objetos que extienden de JpaRepository, proveyendo esta última, acceso a datos ya manejados por el framework. Devuelve objetos Entities.
- **Entities**, es el componente que agrupa los objetos Entity, donde cada uno de estos se mapea contra una tabla en la base de datos.

- **DTOs**, es el componente que agrupa los objetos DTOs, siendo estos los objetos que finalmente son respuesta a los request del cliente desde el Controller.

- **Controllers**, es el componente que agrupa los Controller de la aplicación, siendo estos donde se definen los distintos endpoints de la aplicación. Recibe solicitudes del cliente, y devuelve resultados de esas solicitudes, pudiendo ser respuestas exitosas, erróneas o excepciones. Utiliza DTOs para el flujo de información a través de objetos BusinessServices para resolver las solicitudes. En caso de que ocurran excepciones en tiempo de ejecución, se manejan desde un Controller específico que se encarga de identificarlas, siendo estas excepciones las implementadas en el componente Exceptions.

- **Exceptions**, es el componente que agrupa las excepciones manejadas. Permiten identificar y realizar determinadas acciones frente a un error específico.

Tecnología a utilizar

Para la implementación del API REST se utilizaran:

- Java 11
- Spring Boot version 2.6.7
- Mysql 8.0

Logging

Para la implementación del logger se utiliza la que provee Spring Boot, Logback. El archivo de configuración de este se encuentra en /src/main/resources/logback.xml

NOTA: Se comentan las entradas de log en el código fuente, para poder generar este archivo hay que descomentar las entradas de log y modificar el parámetro de la ubicación del archivo generado en el logback.xml

Testing

Para el testing unitario se utiliza Swagger-UI y Junit 5.

La interfaz de Swagger-UI puede ejecutarse en http://HOST:8080/intraway/swagger-ui/index.html

Es posible realizar un testing manual en Swagger-UI de los tres servicios que expone la API desarrollada. A saber:

1. http://HOST:8080/intraway/api/fizzbuzz/MIN/MAX
2. http://HOST:8080/intraway/api/fizzbuzz/getAllOkResults
3. http://HOST:8080/intraway/api/fizzbuzz/getAllErrorResult

En Junit se implementaron 3 test, a saber:

1. contextLoadsTest() : Test que valida que el controller es generado al cargar el contexto de la aplicación
2. fizzBuzzAlgorithmTest() : Test que permite validar que el algoritmo FizzBuzz implementado funciona correctamente
3. validateResponseContentTypeIsJsonTest() : Test que permite validar que la respuesta del endpoint es un objeto JSON

Al compilar no se ejecutan por defecto los test unitarios (esto se configura en el archivo pom), para ejecutar los test al compilar desde Maven es necesario escribir una bandera de esta forma: mvn -Dtests.skip=false PARAMETRO_MAVEN